Hybrid techniques for classical planning

Nathaniel Waisbrot

Abstract

Two common types of planning systems are "domain-independent" and "domain-configurable". Domain-configurable planners can perform very well, but require much hand-tuning for peak performance. Domain-independent planners do not require human aid, but do not perform as well on some problems. Hybrid planners attempt to combine the strengths of these two styles while minimizing their weaknesses.

Introduction

Consider two general planning strategies: in the first, we gather as much information as possible about the problem, including any known algorithms for solving it, create an informed planner, and then use this planner to solve the problem; in the second, we do not perform any work in advance, and the planner begins trying to compose a solution to the problem with only a shallow understanding of the problem domain. The first strategy belongs to the domain-specific and domain-configurable planners, the second to the domain-independent planners.

Domain-specific planners are built to deal with a specific problem domain or problem. While any algorithm can be considered a domain-specific planner, the Towers of Hanoi puzzle is a typical example of a toy domain used in AI planning. For a Towers problem with three pegs and any number of rings, there exist known algorithms to produce optimal solutions. This is the advantage of domain-specific planners: the quality of the planner's solutions is limited only by that planner's author's knowledge of the problem. Their disadvantage is that they are only as good as their creator's knowledge, and they require all of that knowledge to be encoded into a program.

Domain-independent planners represent the start of Planning as its own field within AI, and they are also the most numerous type of planner. A domain-independent planner requires definitions for any basic action which can be performed in the problem domain, but nothing more. To solve the Towers of Hanoi problem, a domain-independent planner could be given a *move-ring(ring, from, to)* operator, which includes the rules of the puzzle: only the top-most ring on a peg may move, and a larger ring may never be placed on top of a smaller. With this basic information, the planner could attempt to assemble a sequence of *move-ring* actions to produce the solution. In opposition to domain-specific planners, domain-independent planners do not rely on human understanding of the problem or any other external knowledge, so the initial cost is much lower. However, there is no guarantee that such a planner will find a solution rapidly, or that the first solution it finds will be a good one.

Domain-configurable planners are also able to use extensive domain knowledge, but instead of that knowledge being encoded into a programming language, it is encoded into a representation which the planner uses to constrain its domain-independent search. Domain-configurable planners have the most of the same advantages as domain-specific planners, and the same disadvantages, except that behavior common to all planning problems has been abstracted out, reducing the effort needed to plan in a new domain. This abstraction involves some implicit assumptions about the nature of the planning problems, but where these assumptions hold, the performance of the domain-configurable planner can approach that of a domain-specific planner.

There have been a number of techniques suggested to find a middle-ground or hybrid planner between domain-independent planning and domain-configurable planning. This work is based on the hypothesis that, within a planning domain, some portions of plan generation will be easy and well within the capabilities of domain-independent planners, and some portions may be hard and require the deeper domain knowledge used by domain-configurable planners. I will present a survey and comparison of these hybrid planning techniques.

First, I will provide an overview of domain-independent and domain-dependent planning, including examples from two such planners. Next, I will introduce the various planners that hybridize domain-independent and domain-configurable planning or produce a similar effect. Finally, I offer some comparisons of the various systems.

Preliminaries

I provide a short overview of classical planning and some core definitions here, based on those found in *Automated Planning* (Ghallab, Nau, and Traverso 2004)–full details may be found in that text.

A *planning domain* is defined as a triple $\Sigma = (S, A, \gamma)$ where S is the set of states, A is the set of actions, and $\gamma : S \times A \rightarrow S$ is the state-transition function. A given

state s_i contains a set of predicates which hold in that state. A given action a_j is a triple of three subsets, *preconditions, add-effects,* and *delete-effects*. If all formulae in the preconditions of action a_j hold in state s_i, then a_j is *applicable* to state s_i. If an action is applicable, then $\gamma(s_i, a_j)$ will produce some state $s'_i = (s_i \setminus \textit{delete-effects}(a_j)) \cup \textit{add-effects}(a_j)$.

A *planning problem* is a triple $\Pi = (\Sigma, s_0, g)$, where Σ is a planning domain as defined above, s_0 is the initial state of the problem, and g is a set of predicates defining the goal conditions. A planning problem is solved with a plan $\pi = (a_1, a_2, a_3, ...)$ such that if the first action is applied to s_0 and succeeding actions are applied to succeeding states, then the final state will contain g.

For simplicity, I restrict my discussion of planners to those which work in *classical* domains. Classical planning domains satisfy a set of simplifying assumptions based on those made by the original STRIPS planner (Fikes and Nilsson 1971). These assumptions are that S is finite, that Σ is fully observable at all times, that Σ is deterministic, and that plans consist of a linear sequence of actions.

Domain-independent planning

Domain-independent planners make use of general heuristics to search for a sequence of operators connecting the initial planning state to a goal state. The only domain information required by these planners is the set of operators, their preconditions, and their effects (positive and negative).

Nearly all domain-independent planners developed recently take their domain description in Planning Domain Definition Language (PDDL) (Edelkamp and Hoffmann 2004) format, which is based on the domain descriptions used by the STRIPS planner. A PDDL domain description contains: names and types of every individual object which might appear in a planning problem, all predicates which may be used, operator definitions, and axioms used in operator preconditions for convenience.

The FF heuristic

To provide a concrete example of domain-independent planning, I present an overview of the heuristic used by the FF planner (Hoffmann and Nebel 2001). FF combined the novel ideas of GraphPlan (Blum and Furst 1997) with a forward search to produce a very fast and efficient planner.

The FF heuristic provides a lower bound on the number of actions that must be performed to transition from the current state to a goal state. Figure 1 describes a high-level view of the algorithm: we create a "relaxed" set of operators by removing all negative preconditions from all operators, then every applicable operator is applied to the current state and a new pseudo-state is created as the union of all the operator effects and the current state. Figure 2 shows a single step of this process for a domain involving stacking blocks with a robot arm. In the initial state, s, blocks y and z form the tops of two stacks. The robot arm can grab either one, so in the succeeding pseudo-state, s', both blocks have been grabbed. This is a pseudo-state and not a true state because it is not actually possible for the arm to hold two blocks at once, or

function **FF-heuristic**(s, g, O)

1 for each applicable o in O do
2 $s = s \cup \gamma(s, o)$
3 done
4 if $g \subset s$ then return 1
5 else return (1 + FF-distance(s, g, O))

s is the current pseudo-state, g is the set of goal conditions, and O is the set of all operators in the domain with their negative preconditions removed

Figure 1: Pseudocode for the FF heuristic

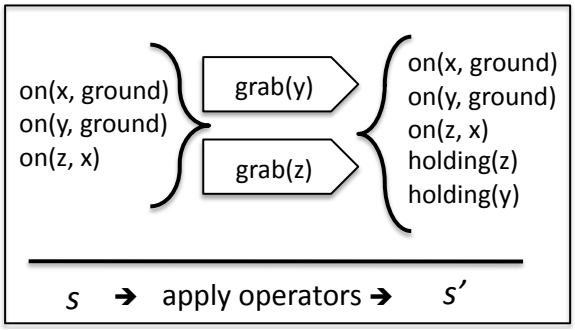

Figure 2: One step of the FF heuristic. Two instances of the *grab* operator are applicable to state s, producing pseudo-state s'.

for blocks to be simultaneously on the ground and held by the arm. The FF-heuristic is repeated until this pseudo-state contains the goal conditions, and the number of iterations is the minimum number of operators which must be applied to reach the goal.

Given any classical problem, the FF heuristic can determine the minimal number of steps between any given state and any given goal, and can also determine if a goal is unreachable (if the pseudo-state does not change after an iteration, then it contains every state which is reachable from the current state). No adjustments are needed to prepare the heuristic for a new problem domain, and the FF planner has been shown to perform well on a variety of planning domains.

However, domain-independent planners like FF do not always perform as well as we might like. FF uses a relaxed set of operators to avoid tracking complex mutex relationships between effects, but this means that more complex domains can puzzle it. Unsolvable problems may appear solvable to the heuristic, because the solution involves an operator with an unsatisfied negative precondition. Problem solutions may be poor and planning slow if there is complex interaction between operators involving the repeated deletion and addition of predicates that appear in the preconditions of other operators. Finally, FF will consider all possible actions at every step of planning, even when some of the actions are clearly bad.

Domain-configurable planning

Domain-configurable planners use large amounts of domain knowledge, provided by some outside source, to restrict their search as much as possible. These planners can eschew complex heuristics to guide their search because the search space has been reduced to such a manageable size. In general, they produce better plans in less time than domain-independent planners. However, their performance is completely reliant on the knowledge they receive. A domain-configurable planner that is given no knowledge, at best, will perform an exhaustive and undirected search of the state space of a problem. The completeness of a domain-configurable planner is dependent on the correctness of its domain knowledge.

There are two main types of domain-configurable planners: *hierarchical task-network* (HTN) planners, and *temporal logic* planners. The background I present here will focus on HTN planners.

HTN planners encode domain knowledge as a set of *tasks*. A task represents a high-level operation to be performed, such as traveling from one point to another using the most efficient means of transportation. During planning, the planner must decompose each task into its components, until eventually every task has been decomposed into a primitive operator. A task may decompose into a sequence of primitive planning operators, like those used in domain-independent planning, or it may decompose into other tasks.

The SHOP2 planner

To provide a concrete example of HTN planning, I describe SHOP2 (Simple Hierarchical Ordered Planner 2) (Nau et al. 2003), a well-known and proven HTN planning system. In SHOP2, a task consists of a head, arguments, and one or more *methods*. A method describes how a task can be performed, *e.g.*:"to travel from A to B, take a taxi from A to the airport, fly, take a taxi from the airport to B." A method can consist of any sequence of primitive operators and non-primitive tasks.

Figure 3 shows a simple example of the kind of domain knowledge used by SHOP2. The domain knowledge explicitly tells the planner which subtasks to perform and under what conditions they may be performed. When the starting location and the destination are close to each other, the planner will *never* consider any action other than direct travel by taxi, and when the two points are distant from each other, the planner will never consider traveling the entire distance by taxi.

Despite the power of domain-configurable planners, there are some drawbacks. Chief among them is the effort required to produce a body of domain knowledge. In figure 3 I do not show definitions for the axioms *short-distance* and *long-distance*, which could be rather complex, depending on the domain. For non-trivial domains, the domain descriptions become extremely large and intricate.

Human authors can carelessly introduce bugs in the domain description, which impact the planners performance on every problem in the given domain. Even when the domain descriptions are correct, to get the maximum benefit of a domain-configurable planner, they must also be well-crafted, requiring time, effort, and skill on the part of the domain writer.

Operator ***take-taxi****(from, to)*
preconditions: at(from)
effects: ¬at(from), at(to)

Operator ***take-plane****(from,to)*
preconditions: at(from), airport(from), airport(to)
effects: ¬at(from), at(to)

Task ***travel****(from, to)*
Method ***travel-by-taxi-only***
preconditions: short-distance(from, to)
subtasks: *take-taxi(from, to)*
Method ***travel-by-taxi-and-plane***
preconditions: long-distance(from, to), airport(a1), short-distance(a1, from), airport(a2), short-distance(a2, to)
subtasks: *take-taxi(from, at)*, *fly(a1, a2)*, *take-taxi(a2, to)*

Figure 3: A simple SHOP2 domain description for a travel domain

Attempts have been made to learn portions of domain knowledge in HTN, by starting with skeletal methods and refining preconditions (Ilghami and Nau 2002; Xu and Muñoz-Avila 2005), or by using traces of experts executing plans (Garland, Ryall, and Rich 2001; Lent and Laird 1999). More recent research has moved towards learning enough knowledge to replace humans as the primary domain writer (Hogg, Muñoz-Avila, and Kuter 2008), but it remains to be seen how closely these attempts can match human-authored domain descriptions for performance.

Partial-knowledge planning

While domain-independent planners perform well on most of the domains used in planning competitions, it is fairly easy to find or create a domain where these planners perform poorly. The Towers of Hanoi puzzle, mentioned above, is a simple example of this. Solving the problem involves stacking and unstacking the same pairs of rings many times over. In most domains, this behavior is not productive, but in this particular domain, it's the only way to solve the problem.

On the other hand, domain-independent planners *are* quite adequate for many problems. For example, the PaintWall domain described by Long and Fox (Long and Fox 2000) is isomorphic to the well-studied Depots domain, and therefore domain-independent planners which can plan well in Depots can plan just as well in PaintWall. However, the operators and state-atoms used in the domain make it appear wholly different from Depots, and a human creating a domain description for a domain-configurable planner would be unlikely to transfer the domain knowledge from Depots into the new domain.

Ideally, a planner should be able to make use of special domain-specific knowledge to solve difficult domains or difficult portions of a domain, and should be able to use general

domain-independent techniques on easier problems, to reduce its need for a large knowledge base. I have divided these techniques into two categories: macro-planners are domain-independent planners which have been augmented by macro-operators–stored sequences of operators; and hybrid planners use both domain-configurable and domain-independent techniques, switching between them as necessary during planning.

Macro-planning

Macro-operators offer a straightforward way to provide a basic degree of domain-specific knowledge to any classical planner. Most research into macros and planning has centered around the best ways to learn macro-operators automatically, to produce a planner that can exploit domain knowledge like a domain-configurable planner without ever needing a human to encode such knowledge. Current macro-planners are not close to this goal, but their ability to outperform non-augmented domain-independent planners in empirical tests demonstrates the utility of automatically learned domain knowledge over no domain knowledge at all.

One of the earliest examples of macro planning was MACROP, an addition to STRIPS by Fikes, Hart, and Nilsson (Fikes, Hart, and Nilsson 1972). The core procedure they describe for generating macro operators appears in every macro-operator paper since. Beginning with a solution plan to some problem in the target domain, they replace the ground atoms with variables to generalize the plan. They apply this same lifting to the preconditions of each operator, propagating constraints backwards through the plan. The lifted plan is stored, with annotations recording the effect that each step of the plan can achieve. Every subsequence of this lifted plan is a potential macro-operator, and the modified STRIPS planner chooses applicable macros during planning. Fikes *et al.* performed experiments to show that their system produced comparable plans in less time than STRIPS on select problems. However, like any learning system, MACROP does not perform well if given a small set of randomly-selected training data, and the number of potential macros is polynomial in the length of the training plans.

Macro-FF (Botea et al. 2005; Botea, Müller, and Schaeffer 2004) is a modern macro-planner, based on the source code of the FastForward planner (Hoffmann and Nebel 2001). While MACROP abstracts its macro-operators by generalizing a solution plan, Macro-FF's CA-ED algorithm generates its macros through static analysis of the domain operators, and by grouping state information into abstract components in the training problems. The macros are then filtered by checking their applicability to a set of solution plans for the target domain. Macro-FF outperformed the FastForward planner by a significant margin on several standard planning domains.

Macros generated by MACROP could, potentially, be quite long, but the CA-ED algorithm fixes the maximum size of a macro to a constant chosen by the authors. Although the maximum size of macros might be more elegantly determined based on the quantity of training data and available storage space, limiting the size of the operators avoids some of the problems of MACROP.

An interesting contribution from Macro-FF was the articulation of a set of five pruning rules to remove less useful or harmful macros: an operator's precondition cannot be negated by the effects of the operators before it (this would create an unsound plan), a macro cannot contain two disjoint sequences of operators which produce the same effect (this implies that at least one set is superfluous), an operator should share at least one atom between its preconditions and the effects of the operator preceding it and between its effects and the preconditions of the operator following it, and finally the scope of a macro's effects are restricted in an attempt to keep them focused. The goal of these rules is to produce fairly short, directed macros that achieve a specific goal, similar to the *tasks* of HTN planners.

Both MACROP and Macro-FF use macros to accelerate the search process by inserting frequently-used sequences of operators into the plan in a single step. Other work, beginning with Minton's MORRIS system (Minton 1990), has looked at macros as a way to store sequences which the planner is unlikely to generate. This type of macro is significantly more difficult both to generate, and to demonstrate as being useful.

Hybrid planners

The core notion behind hybrid planning is to take the structured use of domain knowledge from domain-configurable planners, but permit a relaxed domain definition that does not necessarily contain knowledge for all facets of a domain. In these gaps where knowledge is missing, the planner should use domain-independent techniques. The domain knowledge can be used to optimize known critical paths in the planner's search, or to guide the planner away from action sequences that are known to be useless in that domain.

Hierarchical task-network planning has frequently been chosen to represent the domain-configurable aspect of hybrid planners. This may be because the hierarchical organization of tasks is a reasonably friendly interface between the humans providing knowledge and the planning systems using the knowledge.

Subbarao Kambhampati has performed extensive work in hybrid planning, beginning with a modification to partial-order planners to allow them to perform HTN planning (Kambhampati 1995; Kambhampati, Mali, and Srivastava 1998). Partial-order planners insert actions into a plan and then perform sucessive refinements of the plan to fix an ordering of actions. During refinement steps, actions may be discovered to conflict with the rest of the plan and removed, or new causal links may be satisfied by inserting new actions. The *Refine-Plan-Htn* algorithm allows a partial-order planner to insert HTN tasks as if they were primitive operators, and then performs task-decomposition during the plan-refinement phase, removing the task and inserting its sub-tasks with the same constraints as the parent. A major focus of this work was to use the implementation of HTN planning in a partial-order planning framework to examine the advantages claimed by HTN proponents. Kambhampati concluded that the main advantage of HTN planning was the ability to strongly restrict the planner's search process.

Other researchers have approached the problem from the

other side, adding domain-independent capability to HTN planners. O-Plan (Currie and Tate 1991), an HTN planner, allows a goal to be marked as "unsupervised", signifying that some outside agent (such as a domain-independent planner) will provide the decomposition necessary to achieve the goal. SIPE (Wilkins 1988) has a similar facility. Estlin *et al.* (Estlin, Chien, and Wang 1997) made a more explicit case for this, arguing that high-level goals should be described with HTNs, while low-level goals should be handled by domain-independent planners. They reasoned that the major benefit of HTN-based knowledge was the ease with which it could be reused: high-level instructions for positioning a radar dish are the same regardless of the exact specifications of the dish, so adding a new type to the array is trivial if the low-level commands are generated through domain-independent methods.

McCluskey *et al.* provided empirical results for hybrid planners with *HyHTN* (McCluskey, Liu, and Simpson 2003), a hybrid system which uses an HTN planner to describe high-level goals and the FastForward planner to achieve low-level goals. Testing with *HyHTN* showed that its performance (both in CPU time and in solution length) was comparable to a complete HTN planner, but without the need to describe the domain at the low level. Although the planner was designed to prototype planning applications for the GIPO-II system, the test results imply that it would perform well as a replacement for a complete HTN planner.

The *Duet* planner (Gerevini et al. 2008) combines two state-of-the-art planners, the HTN planner SHOP2 and the domain-independent planner LPG, using a supervisory algorithm to coordinate between them. Like HyHTN and the other HTN-based planners mentioned above, Duet allows the domain writer to omit definitions of low-level tasks. Unlike the other systems, Duet also allows the domain-independent planner to insert HTN tasks into its plan, and have them decomposed on demand by the SHOP2 planner.

Two main arguments in favor of HTN planning that have been advanced are that properly-constructed HTNs drastically reduce the planner's search space to improve performance, and that HTNs provide a friendly and reusable way for humans to describe high-level domain knowledge to the planning system. In a small set of experiments with Duet, the authors found both of these to be the case.

Duet was tested using a domain called "Museums", formed by combining the well-known Depots domain and the Towers of Hanoi puzzle. Museums was created because LPG and other domain-independent planners have a great deal of trouble with the Towers puzzle, but generally perform quite well in the Depots domain. Later experiments were performed with the Storage domain from the International Planning Competition, a more well-known domain. In both sets of tests, the authors found that the domains contained sub-problems which were difficult for the domain-independent planner to solve. When they gave the planner specialized HTNs describing these subsets, without providing domain knowledge for the rest of the domain, Duet's performance exceeded or matched that of an HTN planner with complete domain knowledge.

When the Duet planner was given high-level domain knowledge concerning which goals to complete first, but no low-level knowledge about how to actually complete them, it generally outperformed the LPG planner running on its own, but was not otherwise impressive. Since the test domains were chosen for their difficulty for domain-independent planners, this is likely also the cause for the unimpressive showing of high-level domain knowledge. Even so, the authors found that both the specialized domain knowledge and the high-level domain knowledge were dramatically smaller and less complex than the complete domain knowledge required to plan with the SHOP2 planner alone.

In theory, Duet's mutual planning system could involve a deeply-recursive exchange between the domain-independent and HTN planners, but the authors did not experiment with any domain complex enough to warrant this. Still, the experiments did demonstrate the two hypothesized strengths of HTN planning, the narrow plan search from specialized domain knowledge and the improved ease of describing domain knowledge with a high-level abstraction.

Conclusion

Both domain-independent and domain-configurable planners offer benefits and drawbacks for automating planning. Domain-configurable planners are fast and efficient, but their performance and even completeness is entirely dependent on the quality of domain knowledge which they receive. Domain-independent planners are able to plan in new domains without requiring a large amount of new knowledge, but their generalized approach to problem solving suffers when they are given domains which violate implicit assumptions, and they may fail to find efficient sequences of actions in well-understood domains. Hybrid planners offer a middle-ground between the two, using only as much domain knowledge as the domain writer wishes to provide, and using domain-idependent techniques to solve the remainder of the problem.

References

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–399.

Botea, A.; Enzenberger, M.; Muller, M.; and Schaeffer, J. 2005. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research* 24:581–621.

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Using component abstraction for automatic generation of macro-actions. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, 181–190. Whistler, Canada: ICAPS 2004.

Currie, K., and Tate, A. 1991. O-plan: the open planning architecture. *Artificial Intelligence* 52:49–86.

Edelkamp, S., and Hoffmann, J. 2004. Pddl2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik.

Estlin, T. A.; Chien, S. A.; and Wang, X. 1997. An argument for hybrid htn/operator planning. In *4th European Conference on Planning*.

Fikes, R. E., and Nilsson, N. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 5(2):189–208.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3(4):251–288.

Garland, A.; Ryall, K.; and Rich, C. 2001. Learning hierarchical task models by defining and refining examples. In *In First Int. Conf. on Knowledge Capture*, 44–51. ACM Press.

Gerevini, A.; Kuter, U.; Nau, D.; Saetti, A.; and Waisbrot, N. 2008. Combining domain-independent planning and htn planning: The duet planner. In *18 European Conference on Artificial Intelligence*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning*. New York: Morgan Kaufmann.

Hoffmann, J., and Nebel, B. 2001. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hogg, C.; Muñoz-Avila, H.; and Kuter, U. 2008. Htn-maker: Learning htns with minimal additional knowledge engineering required. In *AAAI*, 950–956.

Ilghami, O., and Nau, D. S. 2002. Camel: Learning method preconditions for htn planning. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, 168–178. AAAI Press.

Kambhampati, S.; Mali, A.; and Srivastava, B. 1998. Hybrid planning for partially hierarchical domains. In *AAAI-98*.

Kambhampati, S. 1995. A comparative analysis of partial order planning and task reduction planning. *SIGART Bulletin, Special Section on Evaluating Plans, Planners and Planning agents* 6(1).

Lent, M. V., and Laird, J. 1999. Learning hierarchical performance knowledge by observation. In *In Proc. 16th Int. Conf. on Machine Learning*, 229–238. Morgan Kaufmann.

Long, D., and Fox, M. 2000. Automatic synthesis and use of generic types in planning. In *Proceedings of AIPS 2000*, 196–205.

McCluskey, T. L.; Liu, D.; and Simpson, R. M. 2003. GIPO II: HTN planning in a tool-supported knowledge engineering environment. In *Proc. of ICAPS-03*.

Minton, S. 1990. Selectively generalizing plans for problem-solving. In Hendler, J. A.; Allen, J.; and Tate, A., eds., *Readings in planning*, Representation and Reasoning. San Mateo, CA: Morgan Kaufmann. chapter 9, 651–654.

Nau, D. S.; Au, T. C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. San Mateo, CA: Morgan Kaufmann.

Xu, K., and Muñoz-Avila, H. 2005. A domain-independent system for case-based task decomposition without domain theories. In *AAAI*, 234–240.